# 会话跟踪技术

HTTP是无状态协议！

在无状态协议下，使用会话跟踪技术来跟踪用户的状态！

会话是什么？

* 会话：会晤！会议！一次谈话！
* B/S会话：
* 参与者：一个客户端和服务器！
* 会话开始：当你使用浏览器打开本站的第一个页面时；

会话结束：关闭浏览器！

## 1.5 会话跟踪常用的方法:

a) URL重写:URL(统一资源定位符)是Web上特定页面的地址，URL重写的技术就是在URL结尾添加一个附加数据以标识该会话,把会话ID通过URL的信息传递过去，以便在服务器端进行识别不同的用户

b)隐藏表单域:将会话ID添加到HTML表单元素中提交到服务器,此表单元素并不在客户端显示

c)Cookie:Cookie是Web服务器发送给客户端的一小段信息，客户端请求时可以读取该信息发送到服务器端，进而进行用户的识别。对于客户端的每次请求，服务器都会将Cookie发送到客户端,在客户端可以进行保存,以便下次使用。

客户端可以采用两种方式来保存这个Cookie对象，一种方式是 保存在 客户端内存中，称为临时Cookie，浏览器关闭后 这个Cookie对象将消失。另外一种方式是保存在 客户机的磁盘上，称为永久Cookie。以后客户端只要访问该网站，就会将这个Cookie再次发送到服务器上，前提是 这个Cookie在有效期内。 这样就实现了对客户的跟踪。

Cookie是可以被禁止的。

d)session:每一个用户都有一个不同的session，各个用户之间是不能共享的，是每个用户所独享的，在session中可以存放信息。

在服务器端会创建一个session对象，产生一个sessionID来标识这个session对象，然后将这个sessionID放入到Cookie中发送到客户端，下一次访问时，sessionID会发送到服务器，在服务器端进行识别不同的用户

Session是依赖Cookie的，如果Cookie被禁用，那么session也将失效

创建Session对象

HttpSession session=request.getSession();

HttpSession session=request.getSession(true);

上面二种创建session的方法效果是一样的。首先获取传递进来的在Cookie中存放的sessionID，再根据sessionID查找session对象，如果没有找到将创建一个新的session对象，并保存到Cookie中发送到客户端。如果找到了，就将session对象赋予给引用

HttpSession session=request.getSession(false);

这种方式创建过程: 首先获取传递进来的在Cookie中存放的sessionID，再根据sessionID查找session对象，如果没有找到将返回null，如果找到了，就将session对象赋予给引用

## 1.1什么是会话跟踪技术

了解会话跟踪技术之前我们先来了解一下几个概念:

### 什么是会话

可以把会话理解为客户端与服务器之间的一次会晤，在一次会晤中可能会包含多次请求和响应。例如你给10086打个电话，你就是客户端，而10086服务人员就是服务器了。从双方接通电话那一刻起，会话就开始了，到某一方挂断电话表示会话结束。在通话过程中，你会向10086发出多个请求，那么这多个请求都在一个会话中。

在JavaWeb中，客户向某一服务器发出第一个请求开始，会话就开始了，直到客户关闭了浏览器会话结束。

### 什么是会话跟踪

对同一个用户对服务器的连续的请求和接受响应的监视

### 为什么需要会话跟踪

浏览器与服务器之间的通信是通过HTTP协议进行通信的，而HTTP协议是”无状态”的协议，它不能保存客户的信息，即一次响应完成之后连接就断开了，下一次的请求需要重新连接，这样就需要判断是否是同一个用户，所以才应用会话跟踪技术来实现这种要求

### 会话跟踪技术介绍

在一个会话的多个请求中共享数据，这就是会话跟踪技术。例如在一个会话中的请求如下：

* + 请求银行主页；
  + 请求登录（请求参数是用户名和密码）；
  + 请求转账（请求参数与转账相关的数据）；
  + 请求信誉卡还款（请求参数与还款相关的数据）。

在这上会话中当前用户信息必须在这个会话中共享的，因为登录的是张三，那么在转账和还款时一定是相对张三的转账和还款！这就说明我们必须在一个会话过程中有共享数据的能力。

**简单介绍:当服务器响应客户端的第一次请求时，将会创建一个新的session对象(该对象实现了HttpSession接口)和一个唯一的ID分配给该请求，以后客户将此会话ID与请求一起传给服务器，此会话ID在后续的请求中会将用户与session对象进行匹配，用于识别不同的客户。**

## 1.2会话路径技术使用Cookie或session完成

我们知道HTTP协议是无状态协议，也就是说每个请求都是独立的！无法记录前一次请求的状态。但HTTP协议中可以使用Cookie来完成会话跟踪！

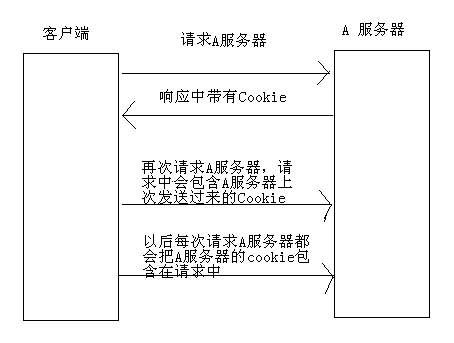
在JavaWeb中，使用session来完成会话跟踪，session底层依赖Cookie技术。下面我们就来认识一下Cookie和Session;

# Cookie

## 2.1 Cookie概述

### 2.1.1　什么叫Cookie

Cookie翻译成中文是小甜点，小饼干的意思。在HTTP中它表示服务器送给客户端浏览器的小甜点。其实Cookie就是一个键和一个值构成的，随着服务器端的响应发送给客户端浏览器。然后客户端浏览器会把Cookie保存起来，当下一次再访问服务器时把Cookie再发送给服务器。



**Cookie是由服务器创建，然后通过响应发送给客户端的一个键值对。客户端会保存Cookie，并会标注出Cookie的来源（哪个服务器的Cookie）。当客户端向服务器发出请求时会把所有这个服务器Cookie包含在请求中发送给服务器，这样服务器就可以识别客户端了！**

**每个域名对应一个Cookie,在设置cookie时需要设定cookie的有效期，以及域名范围，例如你指定到bbs.iteye.com的cookie在访问iteye.com跟域时是不携带的，但是bbs.iteye.com会携带所有iteye.com的cookie，具体的可以详细查看下http协议中cookie部分。 当一个cookie不满足域名限制，或者有效期已经过了，浏览器发的请求就不再携带它，服务器通过判断cookie是否存在，以及存在时是什么值，来做相应的处理**

### 2.1.2　Cookie规范

* Cookie大小上限为4KB；
* 一个服务器最多在客户端浏览器上保存20个Cookie；
* 一个浏览器最多保存300个Cookie；
* 不同浏览器之间不共享Cookie

上面的数据只是HTTP的Cookie规范，但在浏览器大战的今天，一些浏览器为了打败对手，为了展现自己的能力起见，可能对Cookie规范“扩展”了一些，例如每个Cookie的大小为8KB，最多可保存500个Cookie等！但也不会出现把你硬盘占满的可能！

注意，不同浏览器之间是不共享Cookie的。也就是说在你使用IE访问服务器时，服务器会把Cookie发给IE，然后由IE保存起来，当你在使用FireFox访问服务器时，不可能把IE保存的Cookie发送给服务器。

### 2.1.3　Cookie与HTTP头

Cookie是通过HTTP请求和响应头在客户端和服务器端传递的：

* Cookie：请求头，客户端发送给服务器端；
* 格式：Cookie: a=A; b=B; c=C。即多个Cookie用分号离开；
* Set-Cookie：响应头，服务器端发送给客户端；
* 一个Cookie对象一个Set-Cookie：

Set-Cookie: a=A 一个Cookie是一个键一个值。

Set-Cookie: b=B

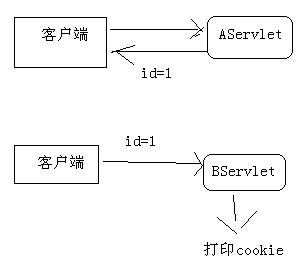
Set-Cookie: c=C

### 2.1.4　Cookie的覆盖

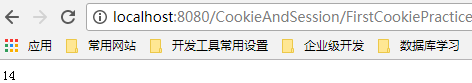
　　如果服务器端发送重复的Cookie那么会覆盖原有的Cookie，例如客户端的第一个请求服务器端发送的Cookie是：Set-Cookie: a=A；第二请求服务器端发送的是：Set-Cookie: a=AA，那么客户端只留下一个Cookie，即：a=AA。

### 2.1.5　Cookie第一例

我们这个案例是，客户端访问AServlet，AServlet在响应中添加Cookie，浏览器会自动保存Cookie。然后客户端访问BServlet，这时浏览器会自动在请求中带上Cookie，BServlet获取请求中的Cookie打印出来。



|  |
| --- |
| package practice.cookie;  import java.io.IOException;  import javax.servlet.ServletException;  import javax.servlet.annotation.WebServlet;  import javax.servlet.http.Cookie;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  /\*\*  \* Cookie的第一个练习:  \* 客户端访问AServlet，AServlet在响应中添加Cookie，浏览器会自动保存Cookie。  \* 然后客户端访问BServlet，这时浏览器会自动在请求中带上Cookie，BServlet获取请求中  \* 的Cookie打印出来。  \* @author Yorick  \*  \*/  public class FirstCookiePractice extends HttpServlet {    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  // //获取Cookie得到访问次数  // String cookie = request.getHeader("Cookie");  // //判断Cookie不是null,并且去除无用空格后不是空字符串  // if(cookie!=null && !cookie.trim().isEmpty()){  // //获取=字符的脚标,截取后面的数字  // int index = cookie.indexOf("=");  // //注意字符串截取包头不包尾  // int count = Integer.parseInt(cookie.substring(index+1));  // //访问次数加1  // count++;  // //把访问次数放到Cookie中,用于记录访问次数  // response.setHeader("Set-Cookie", "count="+count);  // response.getWriter().print(count);;  // }else {  // //说明是第一次访问  // response.setHeader("Set-Cookie", "count=1");  // response.getWriter().print(1);;  // }  //通过上面我们简单实现了访问次数,那么有没有什么简单的实现方法呢,还有每次浏览器返还给我们的Cookie也不止一个,怎么解决呢  //声明一个变量作为访问次数  int count = 1;  //首先获取所有浏览器返还给我们的Cookie  Cookie[] cookie = request.getCookies();  //遍历Cookie,获取指定名称的Cookie  for (Cookie cookie2 : cookie) {//注意如果没有Cookie,那么会在这句报错ava.lang.NullPointerException  if(cookie2.getName().equals("count")){  //声明变量作为访问次数  count = Integer.parseInt(cookie2.getValue());  //访问次数加1  count++;  //跳出循环  break;  }  }  // count有两种可能，如果请求中存在count这个Cookie，我们的int类型的count变量值为cookie的值  // 如果请求中不存在名为count的Cookie，int类型变量count的值为1  /\*  \* 2. 保存cnt到Cookie中，添加到响应中  \*/  // response.setHeader("Set-Cookie", "count="+count);  response.getWriter().println(count);  Cookie cookieNew = new Cookie("count", count+"");  //设计以秒计的存活时间  cookieNew.setMaxAge(20\*20);  response.addCookie(cookieNew);  }  } |



|  |
| --- |
| package practice.cookie;  import java.io.IOException;  import java.util.UUID;  import javax.servlet.ServletException;  import javax.servlet.annotation.WebServlet;  import javax.servlet.http.Cookie;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  /\*\*  \* 给客户端发送Cookie  \* @author Yorick  \*/  @WebServlet("/SetCookie")  public class SetCookie extends HttpServlet {    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  //设置响应类型和编码  response.setContentType("text/html;charset=utf-8");  //获取一个随机字符串,作为Cookie的值  String id = UUID.randomUUID().toString();  Cookie cookie = new Cookie("id", id);  response.addCookie(cookie);  response.getWriter().println("Cookie已经给你发过去了!");  String contextPath = request.getContextPath();  response.sendRedirect(contextPath+"/GetCookie");  }  } |
| package practice.cookie;  import java.io.IOException;  import javax.servlet.ServletException;  import javax.servlet.annotation.WebServlet;  import javax.servlet.http.Cookie;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  /\*\*  \* 获取客户端请求中的Cookie  \* @author Yorick  \*/  @WebServlet("/GetCookie")  public class GetCookie extends HttpServlet {    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  response.setContentType("text/html;charset=utf-8");  //获取Cookie  Cookie[] cookies = request.getCookies();  if(cookies!=null){  for (Cookie cookie : cookies) {  if(cookie.getName().equals("id")){  String id = cookie.getValue();  response.getWriter().println(id);  }else {  response.getWriter().println("没有所找的Cookie!");  }    }  }  }  } |

## 2.2　Cookie的生命

### 2.2.1　什么是Cookie的生命

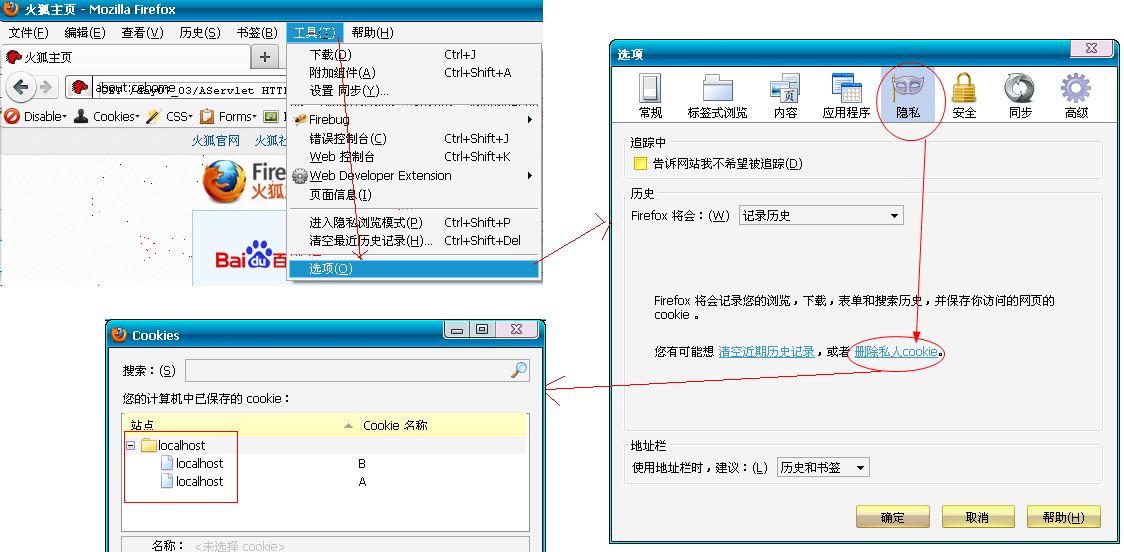
　　Cookie不只是有name和value，Cookie还是生命。所谓生命就是Cookie在客户端的有效时间，可以通过setMaxAge(int)来设置Cookie的有效时间。

* cookie.setMaxAge(-1)：cookie的maxAge属性的默认值就是-1，表示只在**浏览器内存**中存活。一旦关闭浏览器窗口，那么cookie就会消失。
* cookie.setMaxAge(60\*60)：表示cookie对象可存活1小时。当生命大于0时，浏览器会把Cookie保存到硬盘上，就算关闭浏览器，就算重启客户端电脑，cookie也会存活1小时；
* cookie.setMaxAge(0)：cookie生命等于0是一个特殊的值，它表示cookie被作废！也就是说，如果原来浏览器已经保存了这个Cookie，那么可以通过Cookie的setMaxAge(0)来删除这个Cookie。无论是在浏览器内存中，还是在客户端硬盘上都会删除这个Cookie。

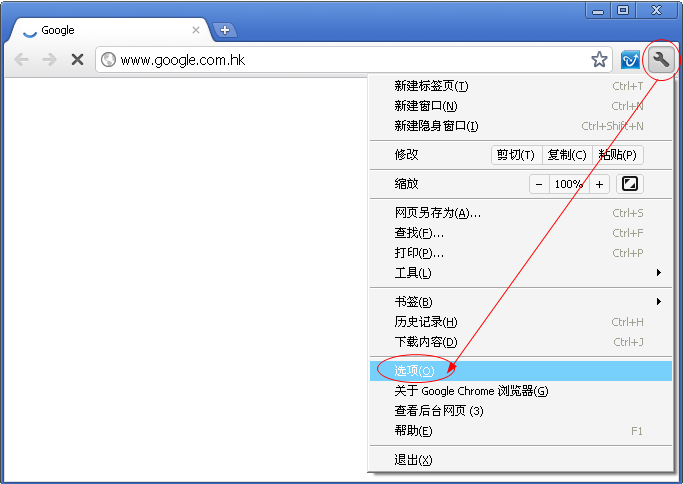
### 2.2.2　浏览器查看Cookie

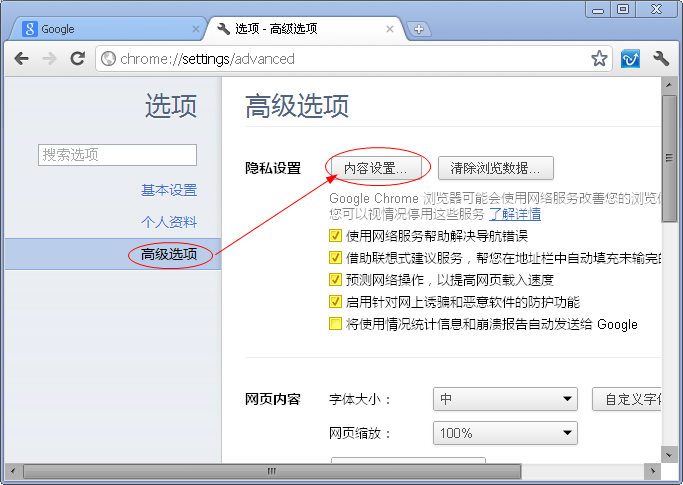
下面是浏览器查看Cookie的方式：

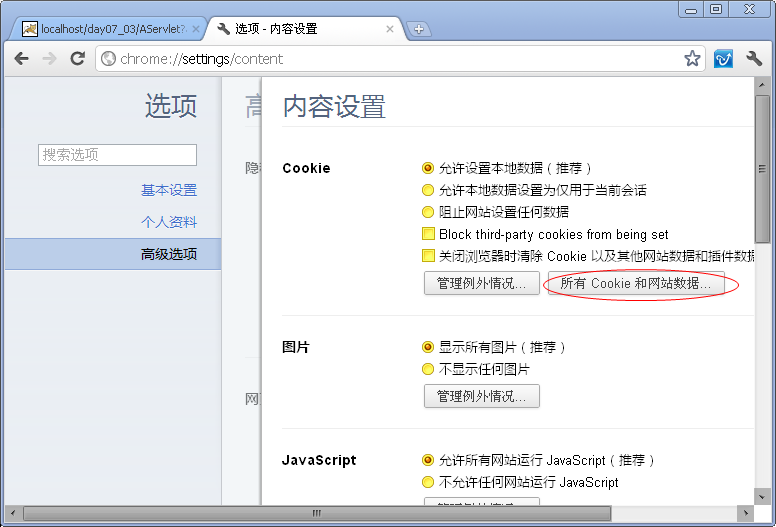
* IE查看Cookie文件的路径：C:\Documents and Settings\Administrator\Cookies；
* FireFox查看Cooke：



* Google查看Cookie：







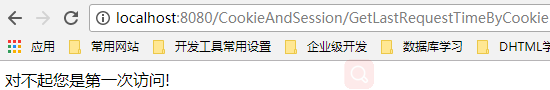


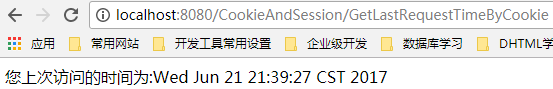
### 2.2.3　案例：显示上次访问时间

* 创建Cookie，名为lasttime，值为当前时间，添加到response中；
* 在AServlet中获取请求中名为lasttime的Cookie；
* 如果不存在输出“您是第一次访问本站”，如果存在输出“您上一次访问本站的时间是xxx”；

AServlet.java

|  |
| --- |
| package practice.cookie;  import java.io.IOException;  import java.util.Date;  import javax.servlet.ServletException;  import javax.servlet.annotation.WebServlet;  import javax.servlet.http.Cookie;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  /\*\*  \* 获取上一次访问的时间  \* @author Yorick  \*创建Cookie，名为lasttime，值为当前时间，添加到response中；  在AServlet中获取请求中名为lasttime的Cookie；  如果不存在输出“您是第一次访问本站”，如果存在输出“您上一次访问本站的时间是xxx”；  \*/  @WebServlet("/GetLastRequestTimeByCookie")  public class GetLastRequestTimeByCookie extends HttpServlet {    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  //设置响应类型和编码  response.setContentType("text/html;charset=utf-8");  //创建Cookie  Cookie cookie = new Cookie("lasttime", new Date().toString());  //设置Cookie的声明周期  cookie.setMaxAge(-1);  response.addCookie(cookie);  //获取Cookie,判断是否为空,获取指定名称的Cookie,获取值  Cookie[] cookies = request.getCookies();  String value = "对不起您是第一次访问!";  if(cookie!=null){  for (Cookie cookie2 : cookies) {  if(cookie2.getName().equals("lasttime")){  value= cookie2.getValue();  value = "您上次访问的时间为:"+value;  //跳出循环  break;  }  }  }  response.getWriter().println(value);  }  } |





### 2.2.4 案例:登录跳转

根据登录是否成功,决定跳转到那个页面,把request域中内容在跳转页面呈现;

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ taglib prefix=*"c"* uri=*"http://java.sun.com/jsp/jstl/core"* %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>My JSP 'login.jsp' starting page</title>  <meta http-equiv=*"pragma"* content=*"no-cache"*>  <meta http-equiv=*"cache-control"* content=*"no-cache"*>  <meta http-equiv=*"expires"* content=*"0"*>  <meta http-equiv=*"keywords"* content=*"keyword1,keyword2,keyword3"*>  <meta http-equiv=*"description"* content=*"This is my page"*>  <!--  <link rel="stylesheet" type="text/css" href="styles.css">  -->  </head>  <body>  <h1>登录页面</h1>  <p style="color:*red*; font-weight: *900*; font-size: *10pt*;">  <%  String msg = (String)request.getAttribute("msg");  **if**(msg != **null**) {  out.print(msg);  }  %>  <%  String username = "";  //request是jsp的内置对象之下，无需创建即可直接使用！  Cookie[] cs = request.getCookies();//获取所有的Cookie  **if**(cs != **null**) {//判断是否存在Cookie，怕出现空指针异常  **for**(Cookie c : cs) {//循环遍历所有的Cookie，查找名为username的Cookie  **if**(c.getName().equals("username")) {  username = c.getValue();//如果查找到，那么把Cookie的值赋给username变量  **break**;  }  }  }  %>  </p>  <form action=*"/CookieAndSession/LoginCheck"* method=*"post"*>  用户名：<input type=*"text"* name=*"username"* value=*"*<%=username %>*"*/>  <input type=*"submit"* value=*"登录"*/>  </form>  </body>  </html> |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <%@ taglib prefix=*"c"* uri=*"http://java.sun.com/jsp/jstl/core"* %>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>My JSP 'succ.jsp' starting page</title>  <meta http-equiv=*"pragma"* content=*"no-cache"*>  <meta http-equiv=*"cache-control"* content=*"no-cache"*>  <meta http-equiv=*"expires"* content=*"0"*>  <meta http-equiv=*"keywords"* content=*"keyword1,keyword2,keyword3"*>  <meta http-equiv=*"description"* content=*"This is my page"*>  <!--  <link rel="stylesheet" type="text/css" href="styles.css">  -->  </head>  <body>  <h1>主页</h1>  <%  String msg = (String)request.getAttribute("msg");  **if**(msg != **null**) {  out.print(msg);  }  %>  </body>  </html> |
| package practice.cookie;  import java.io.IOException;  import javax.servlet.ServletException;  import javax.servlet.annotation.WebServlet;  import javax.servlet.http.Cookie;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  /\*\*  \* 登录校验,同时尝试向Cookie中存储中文  \* @author Yorick  \*  \*/  @WebServlet("/LoginCheck")  public class LoginCheck extends HttpServlet {    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  //设置请求编码和响应类型及编码  request.setCharacterEncoding("utf-8");  response.setContentType("text/html;charset=utf-8");  //获取用户名  String username = request.getParameter("username");  //假如用户名正确,跳转到succ.jsp,否则跳转到login.jsp  if(username.equals("jing")){  request.setAttribute("message", "您输入的用户名或密码有误!");  request.getRequestDispatcher("/login.jsp").forward(request, response);  }else {  //保存登录成功信息  request.setAttribute("msg", "欢迎" + username + "登录本站！");  //把用户名保存到Cookie,让jsp页面获取Cookie中的名字  Cookie cookie = new Cookie("username", username);  cookie.setMaxAge(-1);  response.addCookie(cookie);  request.getRequestDispatcher("/succ.jsp").include(request, response);  }  }  } |

## 2.3　Cookie的path

每个Cookie都有自己的路径！！！路径与是否归还Cookie相关！！！

Cookie的默认路径（在没有设置路径时），是当前Servlet所在路径！！！

http://localhost:8080**/day11\_3/servlet**/BServlet

### 2.3.1　什么是Cookie的路径

现在有WEB应用A，向客户端发送了10个Cookie，这就说明客户端无论访问应用A的哪个Servlet都会把这10个Cookie包含在请求中！但是也许只有AServlet需要读取请求中的Cookie，而其他Servlet根本就不会获取请求中的Cookie。这说明客户端浏览器有时发送这些Cookie是多余的！

可以通过设置Cookie的path来指定浏览器，在访问什么样的路径时，包含什么样的Cookie。

### 2.3.2　Cookie路径与请求路径的关系

下面我们来看看Cookie路径的作用：

下面是客户端浏览器保存的3个Cookie的路径：

a:　/cookietest；

b:　/cookietest/servlet；

c:　/cookietest/jsp；

下面是浏览器请求的URL：

A:　http://localhost:8080/cookietest/AServlet；

B:　http://localhost:8080/cookietest/servlet/BServlet；

C:　http://localhost:8080/cookietest/jsp/CServlet；

* 请求A时，会在请求中包含a；
* 请求B时，会在请求中包含a、b；
* 请求C时，会在请求中包含a、c；

也就是说，请求路径如果包含了Cookie路径，那么会在请求中包含这个Cookie，否则不会请求中不会包含这个Cookie。

* A请求的URL包含了“/cookietest”，所以会在请求中包含路径为“/cookietest”的Cookie；
* B请求的URL包含了“/cookietest”，以及“/cookietest/servlet”，所以请求中包含路径为“/cookietest”和“/cookietest/servlet”两个Cookie；
* C请求的URL包含了“/cookietest”，以及“/cookietest/jsp”，所以请求中包含路径为“/cookietest”和“/cookietest/jsp”两个Cookie；

### 2.3.3　设置Cookie的路径

设置Cookie的路径需要使用setPath()方法，例如：

cookie.setPath(“/cookietest/servlet”);

如果没有设置Cookie的路径，那么Cookie路径的默认值当前访问资源所在路径，例如：

* 访问http://localhost:8080/cookietest/AServlet时添加的Cookie默认路径为/cookietest；
* 访问http://localhost:8080/cookietest/servlet/BServlet时添加的Cookie默认路径为/cookietest/servlet；
* 访问http://localhost:8080/cookietest/jsp/BServlet时添加的Cookie默认路径为/cookietest/jsp；

## 2.4　Cookie的domain（了解一下）

**Cookie的domain属性可以让网站中二级域共享Cookie，次要！**

百度你是了解的对吧！

http://www.baidu.com

http://zhidao.baidu.com

http://news.baidu.com

http://tieba.baidu.com

现在我希望在这些主机之间共享Cookie（例如在www.baidu.com中响应的cookie，可以在news.baidu.com请求中包含）。很明显，现在不是路径的问题了，而是主机的问题，即域名的问题。处理这一问题其实很简单，只需要下面两步：

* 设置Cookie的path为“/”：c.setPath(“/”)；
* 设置Cookie的domain为“.baidu.com”：c.setDomain(“.baidu.com”)。

当domain为“.baidu.com”时，无论前缀是什么，都会共享Cookie的。但是现在我们需要设置两个虚拟主机：www.baidu.com和news.baidu.com。

第一步：设置windows的DNS路径解析

找到C:\WINDOWS\system32\drivers\etc\hosts文件，添加如下内容

|  |
| --- |
| 127.0.0.1 localhost  127.0.0.1 www.baidu.com  127.0.0.1 news.baidu.com |

第二步：设置Tomcat虚拟主机

找到server.xml文件，添加<Host>元素，内容如下：

|  |
| --- |
| <Host name="www.you.com" appBase="D:\Java\Tomcat\apache-tomcat-7.0.78-windows-x86\apache-tomcat-7.0.78\webapps\www"  unpackWARs="true" autoDeploy="true"  xmlValidation="false" xmlNamespaceAware="false"/>  <Host name="news.you.com" appBase="D:\Java\Tomcat\apache-tomcat-7.0.78-windows-x86\apache-tomcat-7.0.78\webapps\news"  unpackWARs="true" autoDeploy="true"  xmlValidation="false" xmlNamespaceAware="false"/> |

第三步：创建Domain项目，

|  |
| --- |
| import java.io.IOException;  import javax.servlet.ServletException;  import javax.servlet.annotation.WebServlet;  import javax.servlet.http.Cookie;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  /\*\*  \* 实现www,baidu.com和www.new.baidu.com共用一个Cookie,一个设置用户名和密码,另一个获取  \* @author Yorick  \*/  @WebServlet("/SetCookieWWW")  public class SetCookieWWW extends HttpServlet {    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  //设置响应类型和编码  response.setContentType("text/html;charset=utf-8");  //获取用户名  String username = request.getParameter("username");  //设置Cookie  Cookie cookie = new Cookie("username", username);  //设置Cookie的domain  cookie.setPath("/");  cookie.setDomain(".you.com");  response.addCookie(cookie);  response.getWriter().write("Cookie已经设置!");  }  } |

把A项目的WebRoot目录复制到D:\Java\Tomcat\apache-tomcat-7.0.78-windows-x86\apache-tomcat-7.0.78\webapps\www目录下，并把WebRoot目录的名字修改为ROOT。

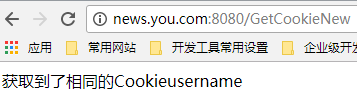
第四步：创建DomainNew项目，

|  |
| --- |
| import java.io.IOException;  import javax.servlet.ServletException;  import javax.servlet.annotation.WebServlet;  import javax.servlet.http.Cookie;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  /\*\*获取Cookie,从Cookie中获取名字,看是否为同一Cookie  \* @author Yorick  \*/  @WebServlet("/GetCookieNew")  public class GetCookieNew extends HttpServlet {    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  response.setContentType("text/html;charset=utf-8");  String mess = "对不起获取到的不是同一Cookie!";  //获取Cookie  Cookie[] cookies = request.getCookies();  if(cookies!=null){  for (Cookie cookie : cookies) {  if(cookie.getName().equals("username")){  mess = "获取到了相同的Cookie"+cookie.getValue();  break;  }  }  }  response.getWriter().write(mess);  }  } |

把B项目的WebRoot目录复制到D:\Java\Tomcat\apache-tomcat-7.0.78-windows-x86\apache-tomcat-7.0.78\webapps\news目录下，并把WebRoot目录的名字修改为ROOT。

第五步：访问www.you.com\SetCookieWWW，然后再访问news.you.com\GetCookieNew。





## 2.5　Cookie中保存中文

　　Cookie中根本不能保存中文！需要把中文转换成URL编码后才能保存。这也说明读取Cookie之后，还要使用URL解码！！！

Cookie的name和value都不能使用中文，如果希望在Cookie中使用中文，那么需要先对中文进行URL编码，然后把编码后的字符串放到Cookie中。

　向客户端响应中添加Cookie

|  |
| --- |
| 使用URL编码  String name = URLEncoder.*encode*("姓名", "UTF-8");  String value = URLEncoder.*encode*("张三", "UTF-8");  编码后的字符串保存到Cookie中  Cookie c = **new** Cookie(name, value);  c.setMaxAge(3600);  response.addCookie(c); |

从客户端请求中获取Cookie

|  |
| --- |
| response.setContentType("text/html;charset=utf-8");  Cookie[] cs = request.getCookies();  **if**(cs != **null**) {  **for**(Cookie c : cs) {  把Cookie的name和value使用URL解码后再打印。  String name = URLDecoder.*decode*(c.getName(), "UTF-8");  String value = URLDecoder.*decode*(c.getValue(), "UTF-8");  String s = name + ": " + value + "<br/>";  response.getWriter().print(s);  }  } |

## 2.6　显示曾经浏览过的商品

index.jsp

|  |  |
| --- | --- |
| <body>  <h1>商品列表</h1>  <a href=*"/day06\_3/GoodServlet?name=ThinkPad"*>ThinkPad</a><br/>  <a href=*"/day06\_3/GoodServlet?name=Lenovo"*>Lenovo</a><br/>  <a href=*"/day06\_3/GoodServlet?name=Apple"*>Apple</a><br/>  <a href=*"/day06\_3/GoodServlet?name=HP"*>HP</a><br/>  <a href=*"/day06\_3/GoodServlet?name=SONY"*>SONY</a><br/>  <a href=*"/day06\_3/GoodServlet?name=ACER"*>ACER</a><br/>  <a href=*"/day06\_3/GoodServlet?name=DELL"*>DELL</a><br/>    <hr/>  您浏览过的商品：  <%  Cookie[] cs = request.getCookies();  **if**(cs != **null**) {  **for**(Cookie c : cs) {  **if**(c.getName().equals("goods")) {  out.print(c.getValue());  }  }  }  %>  </body> | |
| **package practice.cookie;**  **import java.io.IOException;**  **import java.util.Arrays;**  **import java.util.LinkedHashSet;**  **import java.util.Set;**  **import javax.servlet.ServletException;**  **import javax.servlet.annotation.WebServlet;**  **import javax.servlet.http.Cookie;**  **import javax.servlet.http.HttpServlet;**  **import javax.servlet.http.HttpServletRequest;**  **import javax.servlet.http.HttpServletResponse;**  **/\*\***  **\* 获取商品,在index页面打印**  **\* @author Yorick**  **\***  **\*/**  **@WebServlet("/GetGoodsByCookie")**  **public class GetGoodsByCookie extends HttpServlet {**    **protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {**  **//获取请求参数**  **String goodName = request.getParameter("name");**  **//获取Cookie的value**  **String goods = CookieUtils.getCookieValue(request, "goods");**  **if(goods!=null){**  **//按,分割这是Cookie的value的形式[A, p, p, l, e],所以分割和后边的substring是这样的形式**  **String[] arr = goods.split(", ");**  **//将字符串元素放入集合中**  **Set<String> s = new LinkedHashSet<>(Arrays.asList(arr));**  **//把新获取的商品放入集合**  **s.add(goodName);**  **//集合转变成字符串**  **goods = s.toString();**  **goods = goods.substring(1, goods.length() - 1);**  **} else {**  **goods = goodName;**  **}**  **Cookie cookie = new Cookie("goods", goods);**  **cookie.setMaxAge(2\*2);**  **response.addCookie(cookie);**    **response.sendRedirect("/CookieAndSession/index.jsp");**  **}**  **}** | |

CookieUtils

|  |
| --- |
| **package** practice.cookie;  **import** javax.servlet.http.Cookie;  **import** javax.servlet.http.HttpServletRequest;  **public** **class** CookieUtils {  **public** **static** String getCookieValue(HttpServletRequest request,String name){  String value = "";  Cookie[] cookies = request.getCookies();  **if**(cookies!=**null**){  **for** (Cookie cookie : cookies) {  **if**(cookie.getName().equals(name)){  value = cookie.getValue();  **return** value;  }  }  }  **return** **null**;  }  } |

# 3.HttpSession（主菜）

**作用范围**：整个会话范围

**生命周期**：当程序第一次调用request.getSession();方法时，服务器为当前访问的客户端创建对应的Session。

当一个Session超过30分钟（默认时间值，Session的超时时间是可以在web.xml文件中进行配置）没有被操作时，服务器认为该Session超时了，服务器将会销毁Session。

可以调用Session对象提供的invalidate方法，这个Session对象将会自杀，立即失效。

如果web应用被移除出容器，或者服务器被强制停止，随着web应用对应的内存空间被销毁，其中的Session也跟着被销毁了。

如果服务器正常关闭，session将会被钝化到服务器的work目录下的SESSIONS.ser文件中，下次服务器启动讲Session信息还会被活化回内存。

<session-config>

<session-timeout>10</session-timeout>

</session-config>

**共享session:**

利用session原理，自己创建JSESSIONID cookid,并设置maxAge,覆盖自动生成JSESSIONID cookid,使同一台计算机中的多个浏览器可以共用一个session.

**浏览器禁用cooike后的处理办法：**

先利用String urll = response.encodeURL(“带值的路径”);在jsp的链接中加入urll带回。（了解即可 基本不用因为商品太多了不可能一个一个加）

**原理：**

request.getSession();此方法在运行时，会先检查请求消息中有没有JSESSIONID cookie，如果有此cookie则获取这个cookie值作为sesison的id，找到对应的session返回。如果没有此cookie，则会检查访问当前Servlet的URL后面是否跟了JSESSIONID这样一个特殊的属性，如果跟了，获取其值作为session的id，找到对应的session返回。如果既没有JSESSIONID cookie，url后也没跟JSESSIONID，则认为还没有对应的session，创建一个session并返回。创建好session后，在响应消息中设值JSESSIONID cookie。 response.encodeURL（）；判断浏览器是否带了cookie，如果发现浏览器带了cookie则直到浏览器没有禁用cookie，此方法不做任何操作，直接返回传入的URL。 如果发现浏览器没有带着cookie，则认为浏览器可能禁用了cookie，此时会在传入URL后跟上JSESSIONID 这个特殊的属性 浏览器禁用了cookie，将网站中所有的URL都进行重写，使浏览器即使禁用cookie，在访问服务器时仍然可以通过URL带回JSESSIONID的值，从而可以使用SESSION 注意:在进行URL重写之前，需要先创建出session，否则重写无效。

**使用场景:**

1. 使用Session完成用户登录（注销，\*注册）2.利用Session防止表单重复提交

**注意：**

request.getSession()和request.getSession(false);的不同之处：前面的方法一调用，就会去检查是否有对应的session，没有就创建，有就取回。后面的方法只会去检查，如果有就取回，如果没有也不创建。

**域比较：**

**ServeltContext 、reqeust、session域的比较**

servletContext 的作用域是整个web应用，随着服务器启动而创建，如果应用被移除出主机或服务器关闭则销毁。

request 的作用域是整个请求链，每一次请求都会创建一个request，当请求结束时request销毁。

session 的作用于是整个会话，第一次调用reqeust.getSession时创建，当一段时间没有使用或服务器关闭或调用session.invalidate方法时销毁

**什么时候用ServeltContext什么时候用reqeust什么时候用session？**

如果一个数据只是用来显示的话就用request域

如果一个数据除了用来显示以外我一会还要用，这时候用session

如果一个数据除了用来显示以外还要给别人用，这时候用ServletContext域

域对象：

* ServletContext：一个项目只有一个对象；
* HttpSession：一个会话只有一个对象；
* setAttribute()
* getAttribute()
* removeAttribute()
* getAttributeNames()
* HttpServletRequest：一个请求一个对象；

session的特性：

* 它保存在服务器端，浏览器不知道它是什么东西；
* session默认需要Cookie的支持；

得到session：

* servlet：HttpSession session = request.getSession()方法来得到！如果你访问的所有Servlet中都没有调用过这个方法，那么其实session还没有创建！
* jsp：当访问了jsp，那么session就创建了。因为每个jsp的真身中，都会调用request.getSession()。

request.getSession()：

* 查看cookie中的sessionid：如果不存在，创建一个session保存到池中，把sessionid保存到cookie，返回这个新创建的session；
* 查看cookie中的sessionid：如果存在，会到池找到对象的session对象，如果对象存在，那么返回，如果对象不存在，那么还是要创建的；
* request.getSessioni(false)：只会从池中返回session对象，不会去创建！

session作用：

* 用户登录时：保存用户信息；
* 可以使用session保存一次性验证码信息。

## 3.1 HttpSession概述

### 3.1.1　什么是HttpSesssion

javax.servlet.http.HttpSession接口表示一个会话，我们可以把一个会话内需要共享的数据保存到HttSession对象中！

#### session的作用

1. 登录时保存用户信息到session中！

2. 购物车使用需要用到session！

3. 一次性图形验证码！

### 3.1.2　获取HttpSession对象

* HttpSession request.getSesssion()：如果当前会话已经有了session对象那么直接返回，如果当前会话还不存在会话，那么创建session并返回；
* HttpSession request.getSession(boolean)：当参数为true时，与requeset.getSession()相同。如果参数为false，那么如果当前会话中存在session则返回，不存在返回null；

### 3.1.3　HttpSession是域对象

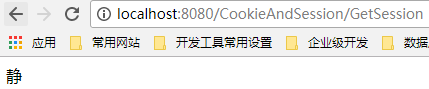
我们已经学习过HttpServletRequest、ServletContext，它们都是域对象，现在我们又学习了一个HttpSession，它也是域对象。它们三个是Servlet中可以使用的域对象，而JSP中可以多使用一个域对象，后面我们再讲解JSP的第四个域对象。

* HttpServletRequest：一个请求创建一个request对象，所以在同一个请求中可以共享request，例如一个请求从AServlet转发到BServlet，那么AServlet和BServlet可以共享request域中的数据；
* ServletContext：一个应用只创建一个ServletContext对象，所以在ServletContext中的数据可以在整个应用中共享，只要不启动服务器，那么ServletContext中的数据就可以共享；
* HttpSession：一个会话创建一个HttpSession对象，同一会话中的多个请求中可以共享session中的数据；

#### session的域方法：

* void setAttribute(String name, Object value)：用来存储一个对象，也可以称之为存储一个域属性，例如：session.setAttribute(“xxx”, “XXX”)，在session中保存了一个域属性，域属性名称为xxx，域属性的值为XXX。请注意，如果多次调用该方法，并且使用相同的name，那么会覆盖上一次的值，这一特性与Map相同；
* Object getAttribute(String name)：用来获取session中的数据，当前在获取之前需要先去存储才行，例如：String value = (String) session.getAttribute(“xxx”);，获取名为xxx的域属性；没有返回null
* void removeAttribute(String name)：用来移除HttpSession中的域属性，如果参数name指定的域属性不存在，那么本方法什么都不做；
* Enumeration getAttributeNames()：获取所有域属性的名称；

|  |
| --- |
| package practice.session;  import java.io.IOException;  import javax.servlet.ServletException;  import javax.servlet.annotation.WebServlet;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  import javax.servlet.http.HttpSession;  /\*\*  \* 向Session中存储数据,重定向到另一个Servlet,获取Session中数据  \* @author Yorick  \*/  @WebServlet("/SetSession")  public class SetSession extends HttpServlet {    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  HttpSession session = request.getSession();  session.setAttribute("username", "静");  response.sendRedirect("/CookieAndSession/GetSession");  }  }  package practice.session;  import java.io.IOException;  import javax.servlet.ServletException;  import javax.servlet.annotation.WebServlet;  import javax.servlet.http.HttpServlet;  import javax.servlet.http.HttpServletRequest;  import javax.servlet.http.HttpServletResponse;  /\*\*  \* 获取Session中数据  \* @author Yorick  \*  \*/  @WebServlet("/GetSession")  public class GetSession extends HttpServlet {    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  String value = (String) request.getSession().getAttribute("username");  response.setContentType("text/html;charset=utf-8");  response.getWriter().write(value);  }  } |



## 3.2登录案例(同Cookie)

需要的页面：

* login.jsp：登录页面，提供登录表单；
* index1.jsp：主页，显示当前用户名称，如果没有登录，显示您还没登录；
* index2.jsp：主页，显示当前用户名称，如果没有登录，显示您还没登录；

Servlet：

* LoginServlet：在login.jsp页面提交表单时，请求本Servlet。在本Servlet中获取用户名、密码进行校验，如果用户名、密码错误，显示“用户名或密码错误”，如果正确保存用户名session中，然后重定向到index1.jsp；

　　当用户没有登录时访问index1.jsp或index2.jsp，显示“您还没有登录”。如果用户在login.jsp登录成功后到达index1.jsp页面会显示当前用户名，而且不用再次登录去访问index2.jsp也会显示用户名。因为多次请求在一个会话范围，index1.jsp和index2.jsp都会到session中获取用户名，session对象在一个会话中是相同的，所以都可以获取到用户名！

login.jsp

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>login.jsp</title>  </head>    <body>  <h1>login.jsp</h1>  <hr/>  <form action=*"/day06\_4/LoginServlet"* method=*"post"*>  用户名：<input type=*"text"* name=*"username"* /><br/>  <input type=*"submit"* value=*"Submit"*/>  </form>  </body>  </html> |

index1.jsp

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>index1.jsp</title>  </head>    <body>  <h1>index1.jsp</h1>  <%  String username = (String)session.getAttribute("username");  **if**(username == **null**) {  out.print("您还没有登录！");  } **else** {  out.print("用户名：" + username);  }  %>  <hr/>  <a href=*"/day06\_4/index2.jsp"*>index2</a>  </body>  </html> |

index2.jsp

|  |
| --- |
| <%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">  <html>  <head>  <title>index2.jsp</title>  </head>    <body>  <h1>index2.jsp</h1>  <%  String username = (String)session.getAttribute("username");  **if**(username == **null**) {  out.print("您还没有登录！");  } **else** {  out.print("用户名：" + username);  }  %>  <hr/>  <a href=*"/day06\_4/index1.jsp"*>index1</a>  </body>  </html> |

LoginServlet

|  |
| --- |
| **public** **class** LoginServlet **extends** HttpServlet {  **public** **void** doPost(HttpServletRequest request, HttpServletResponse response)  **throws** ServletException, IOException {  request.setCharacterEncoding("utf-8");  response.setContentType("text/html;charset=utf-8");  获取表单参数username  String username = request.getParameter("username");  如果用户为itcast表示登录失败  **if**(username.equalsIgnoreCase("itcast")) {  response.getWriter().print("用户名或密码错误！");  } **else** {  获取session对象  HttpSession session = request.getSession();  在session中保存用户名  session.setAttribute("username", username);  重定向到index1.jsp  response.sendRedirect("/day06\_4/index1.jsp");  }  }  } |

## 3.3 session的实现原理

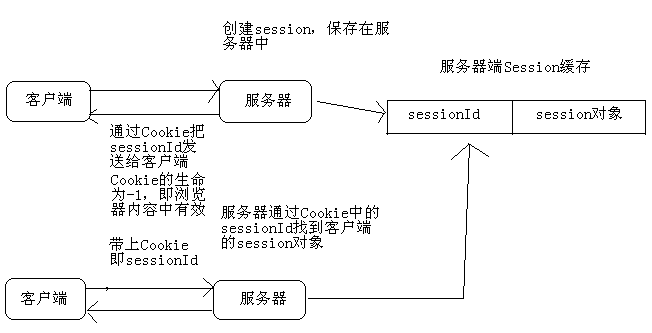
session底层是依赖Cookie的！我们来理解一下session的原理吧！

当我首次去银行时，因为还没有账号，所以需要开一个账号，我获得的是银行卡，而银行这边的数据库中留下了我的账号，我的钱是保存在银行的账号中，而我带走的是我的卡号。

当我再次去银行时，只需要带上我的卡，而无需再次开一个账号了。只要带上我的卡，那么我在银行操作的一定是我的账号！

当首次使用session时，服务器端要创建session，session是保存在服务器端，而给客户端的是session的id（一个cookie中保存了sessionId）。客户端带走的是sessionId，而数据是保存在session中。

当客户端再次访问服务器时，在请求中会带上sessionId，而服务器会通过sessionId找到对应的session，而无需再创建新的session。



## 3.4 session与浏览器

session保存在服务器，而sessionId通过Cookie发送给客户端，但这个Cookie的生命为-1，即只在浏览器内存中存在，也就是说如果用户关闭了浏览器，那么这个Cookie就丢失了。

当用户再次打开浏览器访问服务器时，就不会有sessionId发送给服务器，那么服务器会认为你没有session，所以服务器会创建一个session，并在响应中把sessionId中到Cookie中发送给客户端。

你可能会说，那原来的session对象会怎样？当一个session长时间没人使用的话，服务器会把session删除了！这个时长在Tomcat中配置是30分钟，可以在${CATALANA}/conf/web.xml找到这个配置，当然你也可以在自己的web.xml中覆盖这个配置！

web.xml

|  |
| --- |
| <session-config>  <session-timeout>30</session-timeout>  </session-config> |

session失效时间也说明一个问题！如果你打开网站的一个页面开始长时间不动，超出了30分钟后，再去点击链接或提交表单时你会发现，你的session已经丢失了！

## 3.5 session其他常用API

* String getId()：获取sessionId，其实它就是一个UUID，32位长的字符串；
* int getMaxInactiveInterval()：获取session可以的最大不活动时间（秒），默认为30分钟。当session在30分钟内没有使用，那么Tomcat会在session池中移除这个session；
* void setMaxInactiveInterval(int interval)：设置session允许的最大不活动时间（秒），如果设置为1秒，那么只要session在1秒内不被使用，那么session就会被移除；
* long getCreationTime()：返回session的创建时间，返回值为当前时间的毫秒值；
* long getLastAccessedTime()：返回session的最后活动时间，返回值为当前时间的毫秒值；
* void invalidate()：让session失效！调用这个方法会被session失效，当session失效后，客户端再次请求，服务器会给客户端创建一个新的session，并在响应中给客户端新session的sessionId；

boolean isNew()：查看session是否为新。当客户端第一次请求时，服务器为客户端创建session，但这时服务器还没有响应客户端，也就是还没有把sessionId响应给客户端时，这时session的状态为新。[创建session那个请求中，这个session是新的！]

## 3.6 URL重写

* session默认依赖cookie；
* 浏览器可以禁用cookie，那么session就不能使用了么？

用户访问服务器不过就是通过超链接和表单而已！

* 如果每个请求都有一个名为jsessionid的参数的话，那么也可以不使用cookie。
* 这种方式就是URL重写，但要求每个链接和每个表单都要URL重写。

我们知道session依赖Cookie，那么session为什么依赖Cookie呢？因为服务器需要在每次请求中获取sessionId，然后找到客户端的session对象。那么如果客户端浏览器关闭了Cookie呢？那么session是不是就会不存在了呢？

其实还有一种方法让服务器收到的每个请求中都带有sessioinId，那就是URL重写！在每个页面中的每个链接和表单中都添加名为jSessionId的参数，值为当前sessionid。当用户点击链接或提交表单时也服务器可以通过获取jSessionId这个参数来得到客户端的sessionId，找到sessoin对象。

index.jsp

|  |
| --- |
| <body>  <h1>URL重写</h1>  请求注意，在index.jsp后面使用的是分号，而不是问号，这是服务器对jsessionid这个参数的特殊要求  <a href=*'/day06\_5/index.jsp;jsessionid=*<%=session.getId() %>*'* >主页</a>  <form action=*'/day06\_5/index.jsp;jsessionid=*<%=session.getId() %>*'* method=*"post"*>  <input type=*"submit"* value=*"提交"*/>  </form>  </body> |

也可以使用response.encodeURL()对每个请求的URL处理，这个方法会自动追加jsessionid参数，与上面我们手动添加是一样的效果。

|  |
| --- |
| <a href=*'*<%=response.encodeURL("/day06\_5/index.jsp") %>*'* >主页</a>  <form action=*'*<%=response.encodeURL("/day06\_5/index.jsp") %>*'* method=*"post"*>  <input type=*"submit"* value=*"提交"*/>  </form> |

使用response.encodeURL()更加“智能(该方法会查看cookie是否存在，如果不存在， url重写，否则不重写！)”，它会判断客户端浏览器是否禁用了Cookie，如果禁用了，那么这个方法在URL后面追加jsessionid，否则不会追加。

## 3.7 Session的生命周期

session的生命周期

1. 创建

\* 第一次调用request.getSession()方法或request.getSession(true)！

\* request.getSession(false)：只查找不创建！查找不到返回null

\* jsp的真身中有request.getSession()这句话！

2. 销毁

\* session在长时间不活动才会被销毁！Tomcat中配置是30分钟，可以在${CATALANA}/conf/web.xml找到这个配置，当然你也可以在自己的web.xml中覆盖这个配置！

web.xml

|  |
| --- |
| <session-config>  <session-timeout>30</session-timeout>  </session-config> |

session失效时间也说明一个问题！如果你打开网站的一个页面开始长时间不动，超出了30分钟后，再去点击链接或提交表单时你会发现，你的session已经丢失了！

\* session.invalidate();也可以使session失效！

## 案例：一次性图片验证码

### 1　验证码有啥用

在我们注册时，如果没有验证码的话，我们可以使用URLConnection来写一段代码发出注册请求。甚至可以使用while(true)来注册！那么服务器就废了！

验证码可以去识别发出请求的是人还是程序！当然，如果聪明的程序可以去分析验证码图片！但分析图片也不是一件容易的事，因为一般验证码图片都会带有干扰线，人都看不清，那么程序一定分析不出来。

### 2　VerifyCode类

现在我们已经有了cn.itcast.utils.VerifyCode类，这个类可以生成验证码图片！下面来看一个小例子。

|  |
| --- |
| **package** practice.cookie.image;  **import** java.awt.BasicStroke;  **import** java.awt.Color;  **import** java.awt.Font;  **import** java.awt.Graphics2D;  **import** java.awt.image.BufferedImage;  **import** java.io.IOException;  **import** java.io.OutputStream;  **import** java.util.Random;  **import** javax.imageio.ImageIO;  **public** **class** VerifyCode {  **private** **int** w = 70;//图片的宽度  **private** **int** h = 35;//图片的调试  // 随机对象！  **private** Random r = **new** Random();  // {"宋体", "华文楷体", "黑体", "华文新魏", "华文隶书", "微软雅黑", "楷体\_GB2312"}  **private** String[] fontNames = {"宋体", "华文楷体", "黑体", "微软雅黑", "楷体\_GB2312"};  // 随机生成的字符可选集  **private** String codes = "23456789abcdefghjkmnopqrstuvwxyzABCDEFGHJKMNPQRSTUVWXYZ";    // 它是图片背景色（白色）  **private** Color bgColor = **new** Color(255, 255, 255);  **private** String text ;  **private** Color randomColor () {  /\*  \* 生成随机三元色！  \*/  **int** red = r.nextInt(150);  **int** green = r.nextInt(150);  **int** blue = r.nextInt(150);  // 创建颜色对象返回  **return** **new** Color(red, green, blue);  }  /\*\*  \* 生成随机字体  \* **@return**  \*/  **private** Font randomFont () {  // 生成随机下标  **int** index = r.nextInt(fontNames.length);  // 使用随机下标，得到字体名称  String fontName = fontNames[index];  // 生成一个随机样式  /\*  \* 0 --> 表示不粗不斜  \* 1 --> 粗  \* 2 --> 斜  \* 3 --> 表示又粗又斜  \*/  **int** style = r.nextInt(4);  /\*  \* 生成随机字号  \* 生成0、1、2、3、4，再加上24，最终得到的是24、25、26、27、28  \*/  **int** size = r.nextInt(5) + 24;  // 创建字体对象返回  **return** **new** Font(fontName, style, size);  }  /\*\*  \* 添加干扰线  \* **@param** image  \*/  **private** **void** drawLine (BufferedImage image) {  **int** num = 3;//表示3条  // 获取笔  Graphics2D g2 = (Graphics2D)image.getGraphics();  // 循环画，每次画一条！  **for**(**int** i = 0; i < num; i++) {  /\*  \* 随机获取四个坐标点  \*/  **int** x1 = r.nextInt(w);  **int** y1 = r.nextInt(h);  **int** x2 = r.nextInt(w);  **int** y2 = r.nextInt(h);  /\*  \* 设置笔粗  \*/  g2.setStroke(**new** BasicStroke(1.5F));  /\*  \* 设置线的颜色  \*/  g2.setColor(Color.*BLUE*);    /\*  \* 画线！  \*/  g2.drawLine(x1, y1, x2, y2);  }  }  /\*\*  \* 生成一个随机字符  \* **@return**  \*/  **private** **char** randomChar () {  // r.nextInt(10)，生成0~9之间的一个整数  // 获取字符串的长度为限制！生成的随机数一定是有效下标  **int** index = r.nextInt(codes.length());  // 获取该下标的字符，然后返回  **return** codes.charAt(index);  }  /\*\*  \* 创建空图片  \* **@return**  \*/  **private** BufferedImage createImage () {  // 创建BufferedImage对象  // w、h表示图片的宽和高  BufferedImage image = **new** BufferedImage(w, h, BufferedImage.*TYPE\_INT\_RGB*);  // 获取图片的画笔！  Graphics2D g2 = (Graphics2D)image.getGraphics();  // 设置笔的颜色  g2.setColor(**this**.bgColor);    // 在图片上填充一个矩形, 矩形的左上角在0,0位置上！矩形的宽和高与图片相同！  // 设置图片的背景色为白色！  g2.fillRect(0, 0, w, h);  **return** image;  }  /\*\*  \* 生成图片  \* **@return**  \*/  **public** BufferedImage getImage () {  // 创建图片，但图片上是空的！  BufferedImage image = createImage();  // 获取图片的画笔  Graphics2D g2 = (Graphics2D)image.getGraphics();  // 创建一个Stringbuilder，用来保存随机生成的字符！  StringBuilder sb = **new** StringBuilder();  // 向图片中画4个字符  **for**(**int** i = 0; i < 4; i++) {//循环每次都会向图片上写一个字符！！！  // 生成一个随机字符，转换成字符串  String s = randomChar() + "";  // 把随机生成的字符保存到sb中！  sb.append(s);  // 设置当前字符的x坐标点！  **float** x = i \* 1.0F \* w / 4;  // 设置画笔的字体！  g2.setFont(randomFont());  // 设置画笔的颜色  g2.setColor(randomColor());  // 向图片上画字符  /\*  \* s --> 要画的字符串  \* x --> 字符的x轴坐标  \* h-5 --> 字符的y轴坐标  \*/  g2.drawString(s, x, h-5);  }  // 获取图片上的文本，设置给text！  **this**.text = sb.toString();  // 给图片添加三条干扰线！  drawLine(image);  **return** image;  }  /\*\*  \* 返回图片上的文本！  \* **@return**  \*/  **public** String getText () {  **return** text;  }  **public** **static** **void** output (BufferedImage image, OutputStream out)  **throws** IOException {  ImageIO.*write*(image, "JPEG", out);  }  } |

|  |
| --- |
| **package** practice.cookie.image;  **import** java.awt.image.BufferedImage;  **import** java.io.FileNotFoundException;  **import** java.io.FileOutputStream;  **import** java.io.IOException;  **import** javax.imageio.ImageIO;  **import** org.junit.Test;  **import** cn.itcast.vcode.utils.VerifyCode;  /\*\*  \* 演示VerfiyCode类  \* **@author** Yorick  \*/  **public** **class** VerfiyCodeTest {  @Test  **public** **void** test(){  //获取一个图像验证码对象  VerifyCode vc = **new** VerifyCode();  //创建一张图片  BufferedImage image = vc.getImage();  //获取4个随机字符  String code = vc.getText();  System.*out*.println(code);  //创建输出流,保存图片  **try** {  FileOutputStream os = **new** FileOutputStream("F://image.jpg");  // 使用ImageIO类来完成  // bi --> 要保存的图片  // "JPEG" --> 格式化名称，即什么类型的图片  // out --> 保存到指定的输出流中  ImageIO.*write*(image, "JPEG", os);  } **catch** (FileNotFoundException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  } |

### 3　在页面中显示动态图片

我们需要写一个VerifyCodeServlet，在这个Servlet中我们生成动态图片，然后它图片写入到response.getOutputStream()流中！然后让页面的<img>元素指定这个VerifyCodServlet即可。

|  |
| --- |
| **package practice.cookie.image;**  **import java.awt.image.BufferedImage;**  **import java.io.IOException;**  **import javax.imageio.ImageIO;**  **import javax.servlet.ServletException;**  **import javax.servlet.annotation.WebServlet;**  **import javax.servlet.http.HttpServlet;**  **import javax.servlet.http.HttpServletRequest;**  **import javax.servlet.http.HttpServletResponse;**  **import cn.itcast.vcode.utils.VerifyCode;**  **/\*\***  **\* 生成一张图片，把图片响应给客户端**  **\* @author Yorick**  **\*/**  **@WebServlet("/VCServlet")**  **public class VCServlet extends HttpServlet {**    **protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {**  **//获取一个VerifyCode对象**  **VerifyCode vc = new VerifyCode();**  **//获取图片**  **BufferedImage image = vc.getImage();**  **//获取图片上的文本**  **String code = vc.getText();**  **//把文本放入Session**  **request.getSession().setAttribute("code", code);**  **//把图片输出给客户端**  **response.setContentType("image/jpeg");**  **ImageIO.write(image, "JPEG", response.getOutputStream());**  **}**  **}** |

index.jsp

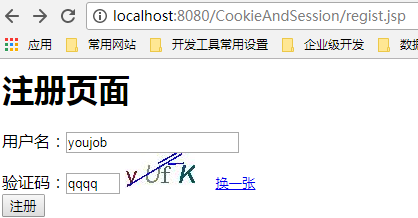
|  |
| --- |
| <script type=*"text/javascript"*>  **function** \_change() {  获取<img>元素  **var** imgEle = document.getElementById("vCode");  重新指定src！因为有些浏览器会对其进行缓存，所以需要添加不同的参数，保证每次URL的参数都不同。  imgEle.src = "/day06\_6/VerifyCodeServlet?" + **new** Date().getTime();  }  </script>  ...  <body>  <h1>验证码</h1>  <img id=*"vCode"* src=*"/day06\_6/VerifyCodeServlet"*/>  点击这个链接会执行\_change()函数  <a href=*"javascript:\_change()"*>看不清，换一张</a>  </body> |

### 4　在注册页面中使用验证码

|  |
| --- |
| <form action=*"/day06\_6/RegistServlet"* method=*"post"*>  用户名：<input type=*"text"* name=*"username"*/><br/>  验证码：<input type=*"text"* name=*"code"* size=*"3"*/>  <img id=*"vCode"* src=*"/day06\_6/VerifyCodeServlet"*/>  <a href=*"javascript:\_change()"*>看不清，换一张</a>  <br/>  <input type=*"submit"* value=*"Submit"*/>  </form> |

### 5　RegistServlet

|  |
| --- |
| **package practice.cookie.image;**  **import java.io.IOException;**  **import javax.servlet.ServletException;**  **import javax.servlet.annotation.WebServlet;**  **import javax.servlet.http.HttpServlet;**  **import javax.servlet.http.HttpServletRequest;**  **import javax.servlet.http.HttpServletResponse;**  **/\*\***  **\* 1. 获取表单中的verifyCode，它是用户填写的验证码**  **\* 2. 获取session中的session\_vcode，它是真实的验证码**  **\* 3. 比较之！**  **\* @author Yorick**  **\*/**  **@WebServlet("/RegistServlet")**  **public class RegistServlet extends HttpServlet {**    **protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {**  **response.setContentType("text/html;charset=utf-8");**  **String mess = "验证码不正确";**  **//获取输入的验证码**  **String verifyCode = request.getParameter("verifyCode");**  **//获取Session中真实的验证码**  **String code = (String) request.getSession().getAttribute("code");**  **//判断**  **if(code.equalsIgnoreCase(verifyCode)){**  **mess = "登陆成功";**  **}**  **response.getWriter().write(mess);**  **}**  **}** |



### 6　总结验证码案例

* VerifyCodeServlet：
* 生成验证码：VerifyCode vc = new VerifyCode(); BufferedImage image = vc.getImage()；
* 在session中保存验证码文本：request.getSession.getAttribute(“vCode”, vc.getText())；
* 把验证码输出到页面：VerifyCode.output(image, response.getOutputStream)；
* regist.jsp：
* 表单中包含username和code字段；
* 在表单中给出<img>指向VerifyCodeServlet，用来在页面中显示验证码图片；
* 提供“看不清，换一张”链接，指向\_change()函数；
* 提交到RegistServlet；
* RegistServlet：
* 获取表单中的username和code；
* 获取session中的vCode；
* 比较code和vCode是否相同；
* 相同说明用户输入的验证码正确，否则输入验证码错误。